Programování v jazyce C pro chemiky
(C2160)

5. Čtení dat ze souboru

Čtení dat ze souboru

```
FILE *f = NULL;
char str[10] = "";
float a = 0.0, b = 0.0;

f = fopen("test1.txt", "r");
if (f == NULL)
{
  printf("Nelze otevrit vstupni soubor!\n");
  return 1;
}

fscanf(f, "%9s %f %f", str, &a, &b);
printf("Nactena data: %s %f %f\n", str, a, b);

fclose(f);
f = NULL;

return 0;
```

Soubor otevíráme v režimu pro čtení **"r"**

Data načítáme funkcí **fscanf()**. Před jména proměnných dáváme znak **&**, kromě řetězcových proměnných.

Vstupní soubor:

```
values 3.78  2.5
```

Čtení dat ze souboru

- Podobné jako při zápisu do souboru
- Soubor otevíráme funkcí **`fopen()`** v režimu pro čtení **`"r"`**
- Testujeme, zdali bylo otevření úspěšné, tj. zdali funkce **`fopen()`** vrátila hodnotu různou od NULL
- Funkce **`fscanf()`** se používá podobným způsobem jako `scanf()`, první parametr je však identifikátor souboru (typu FILE *)
- U jmen proměnných předávaných funkci `fscanf()` musíme používat **`&`** (kromě řetězcových proměnných)

- Pokud neznáme počet hodnot ve vstupním souboru, načítáme zpravidla postupně jednotlivé hodnoty v cyklu, dokud nenarazíme na chybu čtení
- Konec souboru můžeme od ostatních chyb (špatný formát vstupních dat) rozlišit pomocí funkce **`feof(`**`identifikátor souboru`**`)`**, která vrací hodnotu různou od 0 v případě, že předchozí čtení narazilo na konec souboru

Návratová hodnota funkce scanf()

- Funkce `scanf()` a `fscanf()` vrací počet úspěšně načtených hodnot (nezapočítávají se hodnoty s příznakem *, viz dále)
- Návratovou hodnotu využíváme k ověření, zda-li byly správně načteny všechny očekávané hodnoty
- Návratovou hodnotu lze také využít, pokud neznáme přesný počet hodnot, které se na vstupu nacházejí

```
25 186 39
```

```
// Program nacte vyse uvedene hodnoty a vypise, jestli
// se podarilo uspesne nacist vsechny tri
int a = 0, b = 0, c = 0;
int n = 0;

n = scanf("%i %i %i", &a, &b, &c);

if (n == 3)
  printf("Vsechny tri hodnoty byly uspesne nacteny\n");
else
  printf("Bylo nacteno pouze %i hodnot\n", n);
```

Čtení dat ze souboru – příklad 2

```
1  2.5
2  3.5
3  4.7
4  8.6
5  8.9
```

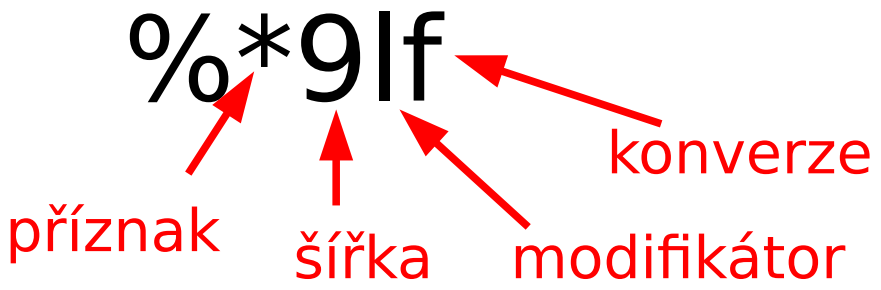
```
// Program nacte soubor s hodnotami (viz. ramecek vlevo)
// Na zacatku programu definujeme #define MAX_ITEMS 100
  int count = 0;                    // Pocet nactenych hodnot
  int numbers[MAX_ITEMS] = {0};     // Pole pro nacteni 1. hodnot
  float values[MAX_ITEMS] = {0.0}; // Pole pro nacteni 2. hodnot
  FILE *f = NULL;

  f = fopen("cisla.txt", "r");
  if (f == NULL) {
    printf("Nelze otevrit vstupni soubor!\n");
    return 1;
  }

  while (fscanf(f, "%i %f", &numbers[count], &values[count]) == 2) {
    count++;
    if (count >= MAX_ITEMS) {
      if (feof(f) == 0) {
        // Jeste nejsme na konci, ale dalsi hodnotu neni kam ulozit
        printf("Velikost pole neni dostatecna!\n");
      }
      break;
    }
  }
  if (feof(f) == 0) {
    printf("Chyba pri cteni souboru po nacteni %i zaznamu!\n",
           count);
  }
  fclose(f);
```

Formátovaný vstup

- Funkce `scanf()` a `fscanf()` používají pro načtení hodnot proměnných formátovací prvky (%i, %f, %c, %s atd.)
- Formátovací prvky specifikují typ proměnné, do které se bude načítat (**`int`**, **`float`**, **`char`**) a zároveň typ dat, která se budou načítat (celé číslo, desetinné číslo, textový řetězec)
- Kromě toho lze ovlivnit způsob načítání pomocí parametrů (příznaky, šířka, modifikátor) – význam těchto parametrů je však odlišný než u funkcí pro formátovaný výstup (printf)
- Obecný zápis formátovacích prvků pro načítání vstupu:

  %[příznaky][šířka][modifikátor]konverze

- Příklad:



Načítání celých čísel - konverze %i a %d

- Konverze **%i** a **%d** slouží pro načítání celých čísel (do proměnné typu **`int`**)
- Konverze %d předpokládá, že číslo je v desítkové soustavě
- Konverze %i předpokládá, že číslo je v desítkové, šestnáctkové (začíná-li číslo znaky 0x nebo 0X) nebo osmičkové soustavě (začíná-li číslo nulou)
- Pokud se před číslem nacházejí mezery, jsou přeskočeny (tj. ignorovány)
- **Šířka** nastavuje maximální počet načtených znaků (do toho však nejsou započítány mezery předcházející před číslem)

Načítání desetinných čísel - konverze %f

- Konverze **%f** slouží pro načítání desetinných čísel (do proměnné typu **`float`**)
- Pokud se před číslem nacházejí mezery, jsou přeskočeny (tj. ignorovány)
- **Šířka** nastavuje maximální počet načtených znaků (do toho však nejsou započítány mezery před číslem)

Načítání řetězce - konverze %s

- Konverze **%s** slouží pro načítání textového řetězce (do pole typu **char []**)
- Tato konverze vynechává všechny počáteční mezery a začne načítat až od prvního znaku, který není mezerou; znaky načítá tak dlouho, dokud nenarazí na mezeru, pak načítání zastaví (tato mezera se již nenačte) – dojde tedy vždy k načtení jednoho slova, které je od následujícího odděleno mezerou
- Před jménem řetězcové proměnné neuvádíme znak **&**
- Po načtení znaků je automaticky vložen zakončovací znak **\0**
- **Šířka** nastavuje maximální počet načtených znaků (do toho však nejsou započítány počáteční mezery) – nakonec se však ještě přidá \0, takže pole musí být dostatečně velké, aby se do něj vešlo *šířka* + *1* znaků
- Šířku téměř vždy uvádíme, abychom zamezili překročení mezí pole při načítání (šířka je o 1 menší než velikost pole)

```
char str[10] = "";       // Pole pro retezec

scanf("%9s", str);  // Nacte max. 9 znaku a na konec vlozi '\0'
```

Formátovaný vstup - konverze %s - příklad

```
Prvni  druhe     treti              slovo
```

```
// Program nacte vyse uvedeny text ze vstupu

char s1[30] = "";
char s2[30] = "";
char s3[30] = "";
char s4[30] = "";

// Pri nacitani se preskakuji pocatecni mezery
// Znaky se potom nacitaji tak dlouho, dokud se nenarazi na mezeru

scanf("%29s", s1);  // Nacte se text: Prvni

scanf("%29s", s2);  // Nacte se text: druhe

scanf("%29s", s3);  // Nacte se text: treti

scanf("%29s", s4);  // Nacte se text: slovo

printf("%s, %s, %s, %s\n", s1, s2, s3, s4);
// Vypise se: Prvni, druhe, treti, slovo
```

Formátovaný vstup - konverze %c

- Konverze **%c** slouží pro načtení znaku (do proměnné typu **char**)
- Tato konverze načítá libovolný znak, tedy i mezeru (na rozdíl od ostatních konverzí, které počáteční mezery vždy vynechávají)
- Není-li specifikována šířka, načítá se jen jeden znak do proměnné `char` (před názvem proměnné musí být **&**)
- **Šířka** umožňuje načíst více než jeden znak, načte se tolik znaků kolik je hodnota šířky; znaky se načítají do pole typu **`char []`**, pole musí mít dostatečnou velikost (minimálně *šířka* + *1*), před názvem pole neuvádíme **&** (protože se jedná o pole)

  POZOR: tato konverze nevloží zakončovací znak \0 (musíme ho tedy vložit sami)

```
char c = ' ';
char s[10] = "";        // Pole znaku

scanf("%c", &c);        // Nacteme znak

scanf("%3c", s);  // Nacteme 3 znaky; pred nazvem pole neni &
s[3] = '\0';      // Vlozime zakoncovaci znak
```

Formátovaný vstup - konverze %[*znaky*]

- Konverze **%[*znaky*]** načítá sekvenci znaků (do pole typu **`char []`**); akceptuje pouze ty znaky, které jsou uvedeny v hranatých závorkách; pokud narazí na znak, který v závorkách uvedený není, načítání se zastaví a dále nepokračuje
- Pokud naopak chceme specifikovat znaky, na nichž se má načítání zastavit, použijeme zápis **%[^*znaky*]**. Tím budou načítány pouze znaky (včetně mezer), které nejsou uvedeny v hranatých závorkách; pokud se narazí na znak, který uvedený je, načítání se zastaví a dále nepokračuje
- Tato konverze nepřeskakuje počáteční mezery
- Po načtení znaků je vložen zakončovací znak \0
- **Šířka** nastavuje maximální počet načtených znaků, nakonec se však ještě přidá \0, takže pole musí být dostatečně velké, aby se do něj vešlo *šířka* + *1* znaků
- Šířku uvádíme téměř vždy, abychom zamezili překročení mezí pole při načítání (šířka je o 1 menší než velikost pole)

Konverze %[*znaky*] - příklad

```
236Cstupnu
```

```
// Program nacte vstup uvedeny vyse
char s1[30] = "";
char s2[30] = "";

scanf("%29[0123456789C]", s1);
printf("Cislo na zacatku: %s\n", s1);   // Vypise: 236C

// Nyni muzeme pokracovat v dalsim nacitani, bude se nacitat
// od mista kde jsme naposledy prestali
scanf("%29s", s2);
printf("Dalsi text: %s\n", s2);   // Vypise: stupnu
```

```
mesic=brezen
```

```
// Program nacte vstup uvedeny vyse
char s1[30] = "", s2[30] = "", s3[30] = "";

// Nacitame vse dokud nenarazime na '=' nebo mezeru
scanf(" %29[^= ]", s1);  // Nacte se: mesic
// Nacitame vsechny znaky '=' nebo mezery
scanf(" %29[= ]", s2);   // Nacte se: =
scanf(" %29s", s3);      // Nacte se: brezen

// Zkracene by se to dalo napsat nasledovne:
// scanf("%29[^= ] %29[= ] %29s", s1, s2, s3);
```

Formátovaný vstup – příznak *

- Při načítání dat pomocí formátovacích prvků lze použít formátovací příznak *
- Tento příznak způsobí, že se daná hodnota načte, ale neuloží se do žádné proměnné
- Tento příznak používáme, pokud některou z hodnot nepotřebujeme zpracovávat, ale musíme ji načíst, aby bylo možné načítat hodnoty za ní

```
mesic=brezen
```

```
// Program nacte vstup uvedeny vyse
// Podobny program jako na predchozi strance, ale tentokrat
// rovnitko nebudeme ukladat to promenne ale zahodime ho
char s1[30] = "", s2[30] = "";

// Nacte se prvni reetzec do promenne s1,
// pak se nacte = s pripadnymi mezerami a zahodi se,
// nakonec se nacte posledni retezec do promenne s2
scanf(" %29[^= ] %*[= ] %29s", s1, s2);

printf("Text pred rovnitkem: %s\n", s1);   // Vypise: mesic
printf("Text za rovnitkem: %s\n", s2);     // Vypise: brezen
```

Formátovaný vstup - znaky ve formátovacím řetězci

- Ve formátovacím řetězci funkcí `scanf()` a `fscanf()` zpravidla uvádíme pouze formátovací prvky
- Pokud však v těchto funkcích uvedeme libovolné jiné znaky, musíme je na vstupu zadat přesně, jak je uvedeno
- Pokud funkce narazí na odlišný znak než je uvedeno, načítání bude neúspěšné (zastaví se na tomto znaku)

```
Cislo je 853
```

```
// Program nacte vstup uvedeny vyse
int a = 0;

scanf("Cislo je %i", &a);

// Pokud se text bude lisit, bude nacteni neuspesne, napr.:
scanf("cislo je %i", &i);  // Prvni pismeno je male
scanf("Cislo %i", &i);     // Nacte text "Cislo", pak se pokusi
   // nacist hodnotu, ale narazi na znak 'j', coz je neplatne
   // pro nacitani celych cisel (ocekava se cifra nebo znamenko)
scanf("%i", &i);  //Pokusi se nacitat cele cislo, ale narazi na
    //znak 'C', coz neni platny znak pro nacitani celeho cisla
```

Formátovaný vstup - mezery ve formátovacím řetězci

- Mezera ve formátovacím řetězci je interpretována jinak než ostatní znaky
- Mezera ve formátovacím řetězci představuje libovolný (i nulový) počet mezer na vstupu, které se mají ignorovat (přeskočit)
- Za „mezeru“ se považují i libovolné jiné „bílé“ znaky (*white-space*): tabelátor, nový řádek, atd.
- Ve formátovacím řetězci dáváme mezery všude tam, kde předpokládáme, že může uživatel vložit jednu nebo více mezer

```
1, 2
3 , 4
5  ,    6
7,8
```

```
// Program nacte libovolny z vyse uvedenych radku
int a = 0, b = 0;

scanf("%i , %i", &a, &b);

// Nasledujici kod by spravne nacetl jen prvni a posledni radek
scanf("%i, %i", &a, &b);
```

Dodržujte následující pravidla

- Velikost polí specifikujte pomocí symbolické konstanty (definované pomocí `#define`). Výjimkou jsou řetězcové proměnné, zejména pokud budou použity pro načítání pomocí funkcí `scanf()` a `fscanf()` a bude tedy nutné specifikovat max. počet načtených znaků ve formátovacím řetězci.
- Hodnoty polí na počátku inicializujte hodnotou 0. Stačí inicializovat první prvek pole, překladač pak všechny ostatní inicializuje nulami
- Řetězcové proměnné inicializujte prázdným řetězcem
- Definice proměnných opatřete komentářem
- Při použití `scanf()` a `fscanf()` nezapomeňte dát & před název proměnné (i pokud se jedná o prvek pole). Výjimkou jsou řetězcové proměnné, kde se & nepoužívá.
- Ve volání funkce `fopen()` nepoužívejte absolutní cesty. Zpracovávané soubory si napřed ručně zkopírujte do adresáře s programem a ve volání `fopen()` používejte již jen jméno souboru bez cesty.

Úlohy – část 1

1. Vytvořte program který načte soubor `data1.dat` (nacházející se v adresáři `/home/tootea/C2160/data/`). Tento soubor obsahuje data týkající se komplexů HIV-1 proteázy s jejími inhibitory. Soubor obsahuje na každém řádku: PDB kód komplexu, experimentální vazebné energie, spočítané vazebné energie. Program potom vypíše na obrazovku hodnoty energií (nikoli PDB kód) v obráceném pořadí řádků (tj. obě hodnoty z posledního řádku se zapíší na první řádek). Hodnoty se budou vypisovat s přesností na 2 desetinná místa. **1 bod**
2. Úlohu 1 upravte tak, že hodnoty nebudou vypisovány na výstup, ale budou zapisovány do souboru. Hodnoty energií formátujte tak, aby byly vypsány s přesností na 2 desetinná místa a desetinné tečky byly zarovnány pod sebou. **1 bod**
3. Vytvořte program, který načte soubor obsahující libovolný počet (omezený vhodnou symbolickou konstantou) celých čísel, oddělených čárkami (počítejte s možností výskytu mezer před a za čárkami). Na konci se vypíše počet těchto hodnot a hodnoty se vypíší v obráceném pořadí. (Nápověda: V cyklu načítejte hodnoty po jedné, dokud funkce fscanf() nevrátí hodnotu různou od 1.) **1 bod**

```
12, 13 , 14,15, 16, 17
```

Úlohy – část 2

4. Vytvořte program, který načte soubor, kde na prvním řádku bude text: "INPUT_FILE = crambin.pdb" a na druhém řádku text: "WINDOW_SIZE: 300, 500". Program načte text před dvojtečkou a rovnítkem do řetězcových proměnných a hodnoty do proměnných typu int a jméno do druhé řetězcové proměnné. Program vypíše hodnoty rozměrů a zadané jméno. Počítejte s možností mezer na začátku každého řádku a před nebo po '=' a ','. (Načítání nyní nebude vyžadovat cyklus, pouze zavoláme dvakrát fscanf(), pro každý řádek jednou). **1 bod**

```
INPUT_FILE = crambin.pdb
WINDOW_SIZE: 300, 500
```

```
Prvni radek, text pred rovnitkem: INPUT_FILE
Prvni radek, text za rovnitkem: crambin.pdb
Druhy radek, text pred dvojteckou: WINDOW_SIZE
Druhy radek, hodnoty: 300, 500
```

5. Vytvořte program který načte soubor *numbers1.dat* a *numbers2.dat* (nacházející se v adresáři /home/tootea/C2160/data/). Program bude porovnávat celočíselné hodnoty na jednotlivých řádcích a do jiného souboru zapíše vždy "identical" nebo "different". Program musí data zpracovávat bez ukládání hodnot do polí. **nepovinná, 2 body**

```
 36
 -5
 21
174
 94
 86
-23
```

```
 36
 50
 21
174
125
 78
-23
```

```
identical
different
identical
identical
different
different
identical
```